

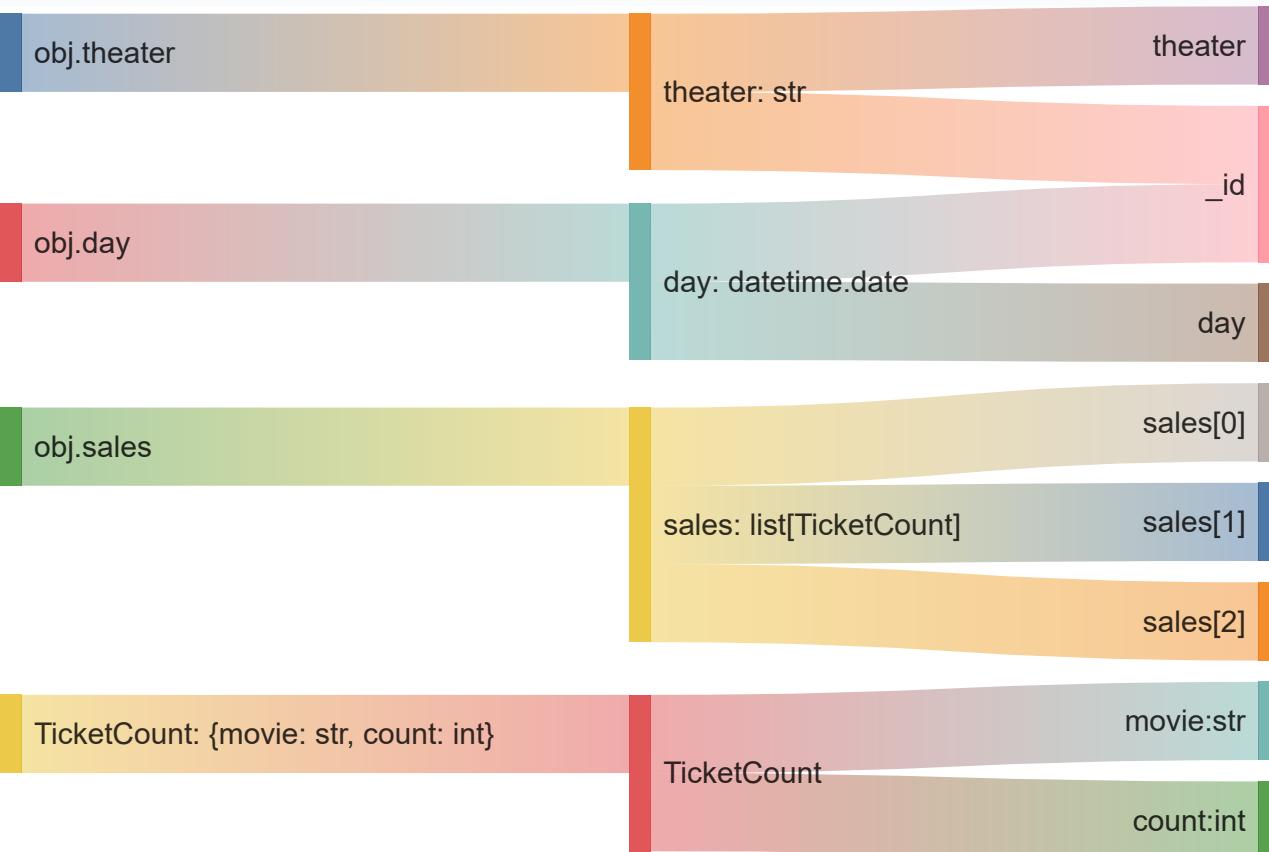
Data How You Want It

PYMONGO + MONGODB == YOUR_DATA_BACKEND

```
discussion_topics = ('MongoDB', 'pymongo', 'FastAPI')
```

Nuri Halperin | PyCon 2025

Pyadantic



A Model

```
class TheaterSales(BaseModel):  
    theater: str  
    day: datetime.date  
    sales: list[TicketCount] = []  
  
    @computed_field(alias="_id")  
    @property  
    def id(self) -> str:  
        return create_id(self.theater, self.day)  
  
    model_config = {  
        "json_schema_extra": {}  
    }
```

A Document

MongoDB stores documents.

Documents are self describing representation of your data.

```
{  
  "theater": "Aero",  
  "day": { "$date": "2025-01-01" },  
  "sales": [  
    { "movie": "Elf", "tickets_sold": 111 },  
    { "movie": "Die Hard", "tickets_sold": 222 }  
  ]  
}
```

NUMBER OF TICKETS SOLD FOR A MOVIE IN A
THEATER ON A CERTAIN DAY._

BSON ~= JSON

JSON is nice, but skimps on native data types.

Field	BSON Data Type	Python (in demo)
theater	UTF-8 String	str
day	UTC Datetime	datetime.date
sales	Array	list
count	32-bit integer	int

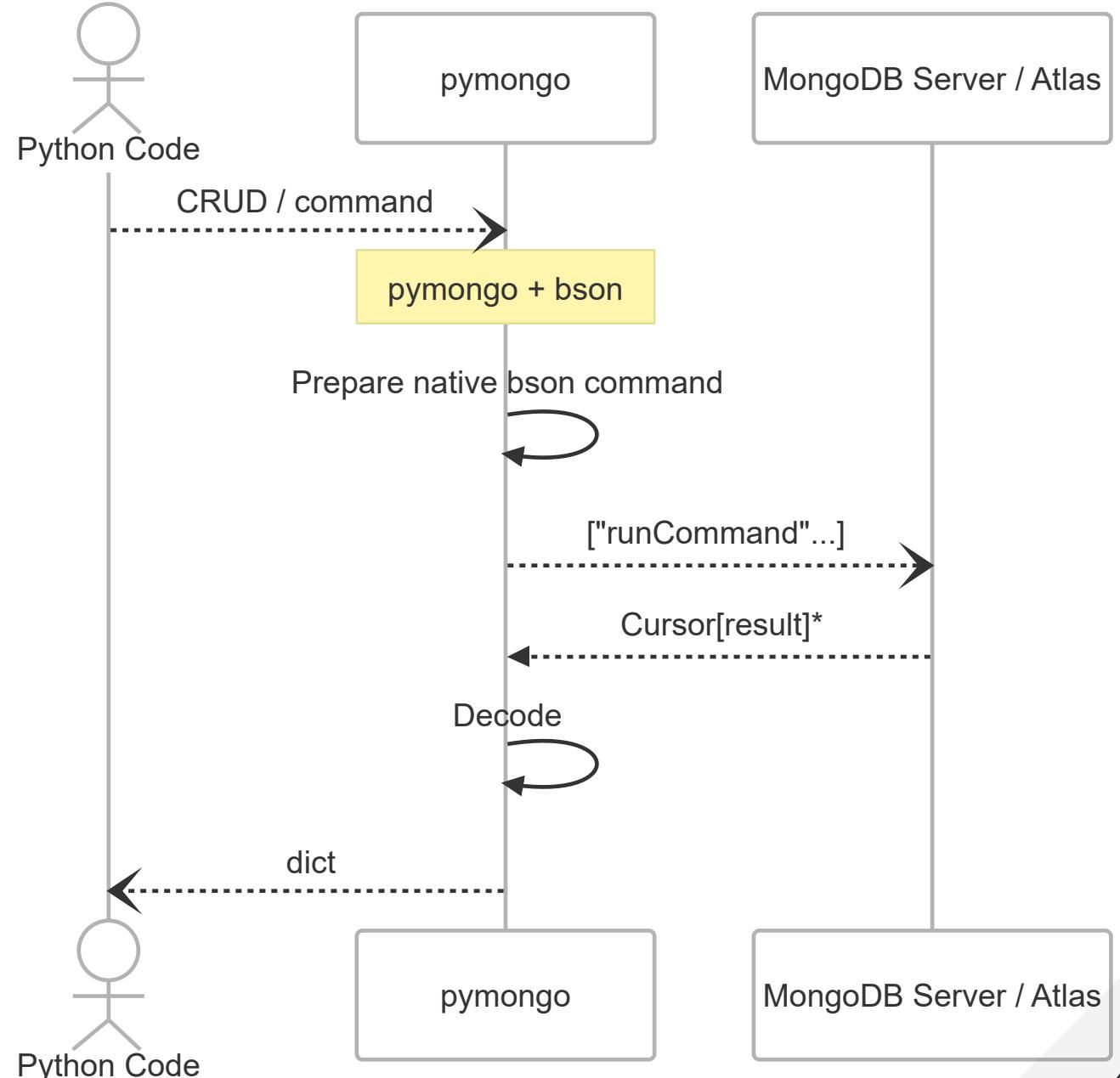
SEE [HTTPS://BSONSPEC.ORG](https://BSONSPEC.ORG)

“

”

PyMongo - Driver

- BSON, codec
- Sync / Async
- Cursor
- Connection management





Diving In

- Instantiation scope
- "Connect"
- "Namespace"

Once, Before All

```
from pymongo import AsyncMongoClient  
  
CONNECTION_STRING = os.environ.get("MY_MONGODB_URI", "mongodb://localhost/demo")
```

INSTANTIATE

```
client = AsyncMongoClient(CONNECTION_STRING)  
db = client.get_default_database()
```

DO SOMETHING...

```
c1 = db.stuff1  
  
# c2 = db.get_collection("stuff1")
```





Read A Doc

```
async def get_one_theater_sales(day: date, theater: str):  
    """Get the sales for a single day."""  
  
    filter = {"_id": f"{day}_{theater.lower().replace(' ', '_')}"}  
  
    doc = await db.theater_sales.find_one(filter)  
    return doc if doc else None
```

“ FIND_ONE() RETURNS ONE DOCUMENT MATCHING CRITERIA. ”

Writing - (Dangerously)

```
await collection.update_one(  
    {"_id": data.id},  
    {  
        "$set": {  
            "sales": sales, # a list of sales  
            "theater":data.theater,  
            "day": date_to_datetime(data.day),  
        }  
    },  
    upsert=True,  
)
```

Writing Respectfully

```
await db.theater_sales.update_one(  
    {"_id": data.id},  
    {  
        "$set": {"theater": data.theater, "day": date_to_datetime(data.day)},  
        "$push": {"sales": {"$each": [m.model_dump() for m in data.sales]}},  
    },  
    upsert=True,  
)
```

- Previous elements of the `sales` field survive

Aggregate - Read More Docs

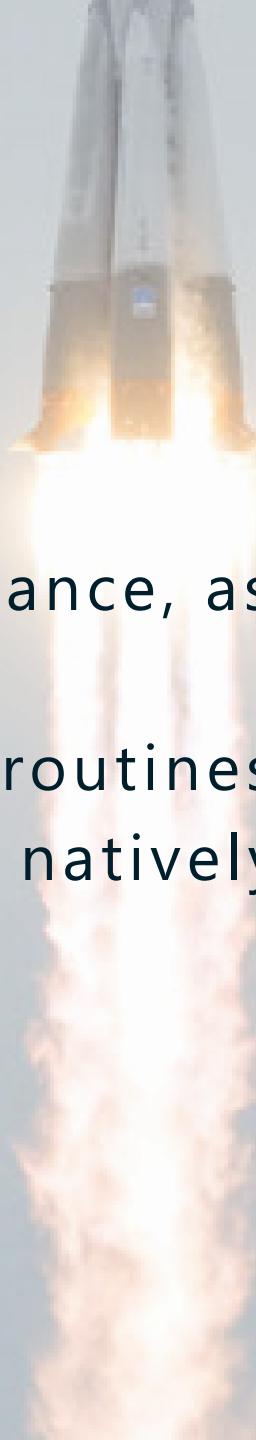
```
async def multi_day_sales(on_or_after: date, before: date, *breakdown: str):
    filter = date_filter(on_or_after, before)
    unwind = unpivot_sales()
    group = group_by(breakdown)
    flatten = flatten_id()
    pipeline = [filter, unwind, group, flatten]

    return await db.theater_sales.aggregate(pipeline).to_list(None)
```

PYMONGO **CURSOR** CAN BE ITERATED, AND HIDES BATCHING DETAILS

Fast?

- FastAPI: Use for high performance, async API
- Pydantic: Validate dict / doc
- Async: AsyncMongoClient (coroutines).
- MongoDB: Sharding, indexes, natively document oriented



Resources

SECTIONS AND DETAILS

Scales : Eight Feet to One Inch
One Inch

1. pymongo pymongo.readthedocs.io/en/stable/
2. AsyncMongoClient [/en/stable/api/pymongo/asynchronous](https://pymongo.readthedocs.io/en/stable/api/pymongo/asynchronous.html)
3. fastapi fastapi.tiangolo.com/
4. MongoDB Community Champions
mongodb.com/community/champions
5. Demo Repo github.com/nurih/data-how-you-want-it